

Lecture 3: Constraint Satisfaction

September 9th, 2026



Gabriel Poesia

Assistant Professor, CSE

University of Michigan, Ann Arbor

poesia@umich.edu

<https://gpoesia.com>

0. Administrivia
1. Factored state representation
2. Constraint satisfaction problems
3. Constraint propagation
4. Backtracking search

ADMINISTRIVIA

Homework 1 will be posted tonight! Due 2 weeks from now (Sep 23rd, 11:59pm)

4 problems:

1. Search (last lecture)
2. CSPs (this lecture)
3. Propositional Logic (next lecture, Monday)
4. First-Order Logic (next Wednesday)

(1), (2) and (3) have a coding component (you will write < 100 lines total)

Two separate deliverables on Gradescope, for written and coding parts.

Written part: we give you a LaTeX template. Submit a PDF.

Gradescope will ask if you want to assign pages to items after you upload your PDF. **Please do this!**

Coding part: upload the zip generated by the script we provide.

Where we left off

A search problem gives us:

- Atomic states $s \in \mathcal{S}$
- Actions $A(s)$
- Transitions $T(s, a)$
- Action costs and a goal test

Uninformed search: rank frontier by node depth or path cost

Informed search / A*: also uses heuristic $h(s)$ to estimate cost to go

With atomic states, the search algorithm does not see what is in s (h does)

Today, we will assume some more structure in the state

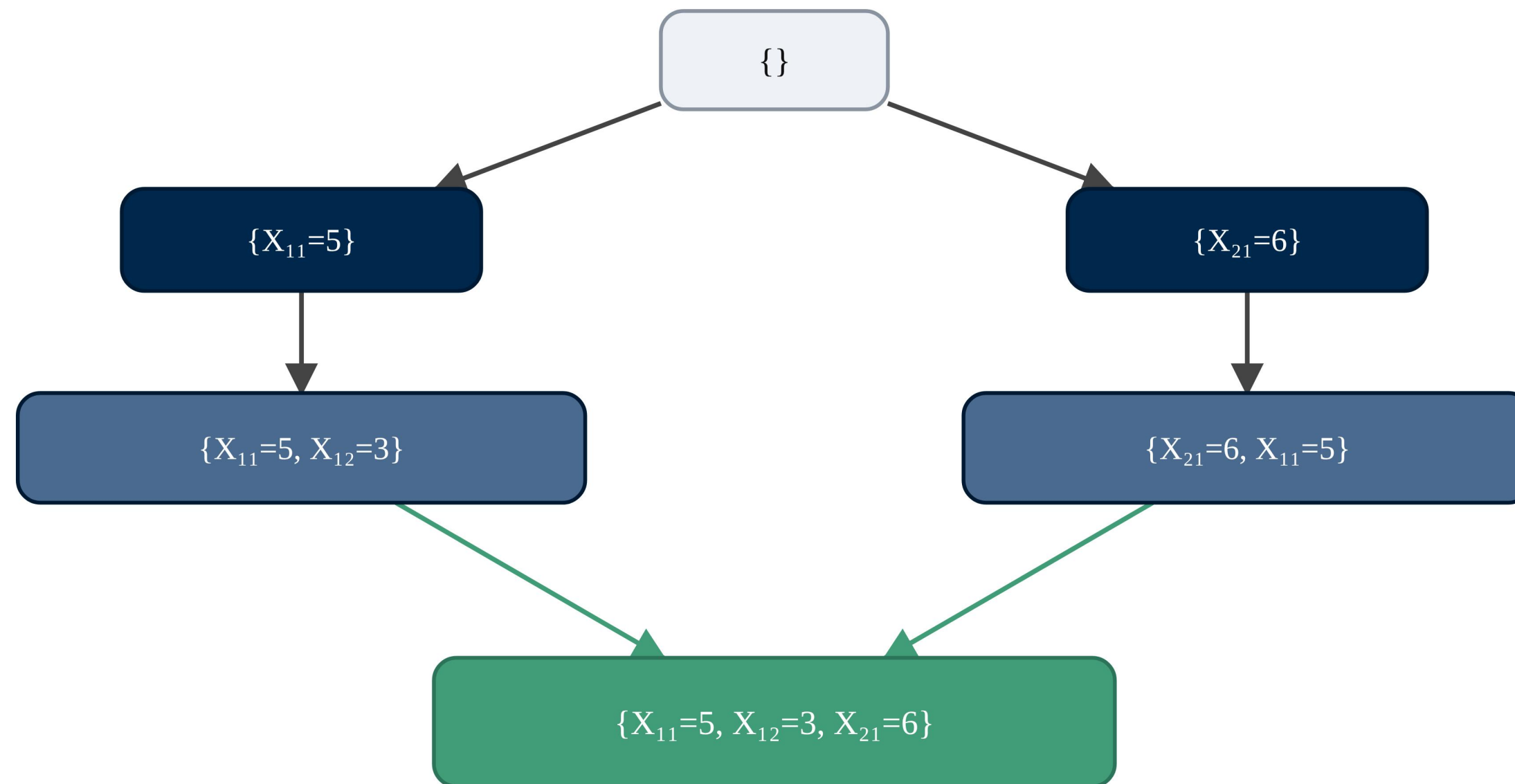
FACTORED STATE REPRESENTATION

UNIVERSITY OF
MICHIGAN

- | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Many roads lead to Rome

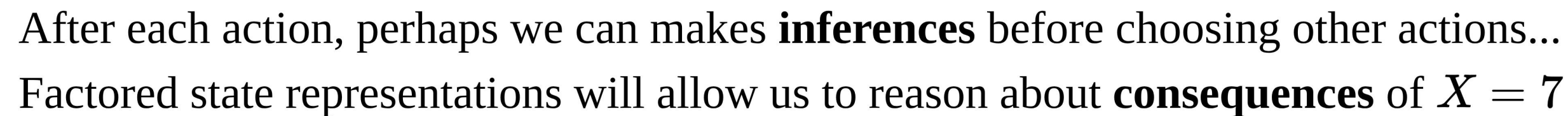
Let X_{rc} be the digit in row r , column c .
Consider these two search branches:



A partial assignment with k entries can be reached in $k!$ orders

Generic search would fail to see this symmetry

Regular search algorithm (e.g., BFS) might make the following choices:



From atomic to factored states

Entire grid

Atomic

$X_{11} = 5$

$X_{12} = 3$

$X_{13} = ?$

$X_{21} = 6$

$X_{22} = ?$

$X_{23} = ?$

...

$X_{31} = ?$

$X_{32} = 9$

$X_{33} = 8$

Factored

DEFINITION (FACTORED STATE REPRESENTATION)

In a **factored** state representation, each state is composed of a collection of **features**.

In Sudoku and many related problems, these features will consist of variable assignments

CONSTRAINT SATISFACTION PROBLEMS

Constraint satisfaction problems

DEFINITION (FINITE-DOMAIN CSP)

A **finite-domain constraint satisfaction problem** (CSP) is a tuple (X, D, C) : $X = \{X_1, \dots, X_n\}$ is a set of variables, $D = \{D_1, \dots, D_n\}$ gives each X_i a finite domain, and $C = \{C_1, \dots, C_m\}$ is a set of constraints.

To specify the problem, we decide:

X : decisions

What decisions do we need to make? (e.g., digit for cell X_{11})

D : options

What options do we have in each decision (e.g., digits 1 – 9)

C : interactions

What combinations are allowed/forbidden? (e.g., $X_{11} \neq X_{12}$)

DEFINITION (CONSTRAINT)

A constraint $C_k = (scope_k, rel_k)$ consists of a **scope** (the collection of variables C_k involves), and a **relation** with the tuples of values those variables are allowed to take.

EXAMPLE: CONSTRAINT $X < Y$

Assuming $D_X = D_Y = \{1, 2, 3\}$, a constraint $X < Y$ would have $scope = (X, Y)$, $rel = \{(1, 2), (1, 3), (2, 3)\}$

Equivalently, rel can be a function taking values for the variables and returning true/false for allowed/disallowed combinations

Assignments and solutions

(X, D, C) specify the **problem**. Key data in the search state is an **assignment**:

DEFINITION (ASSIGNMENT)

An **assignment** is a set $a = \{X_i = v_i, \dots\}$ with $v_i \in D_i$ and at most one value per variable. Writing $\text{dom}(a)$ for the assigned variables, a is **complete** if $\text{dom}(a) = X$, and **partial** otherwise (at least one unassigned variable).

DEFINITION (CONSISTENCY AND SOLUTION)

An assignment a is **consistent** if every constraint whose scope is contained in $\text{dom}(a)$ allows the values in a .
A **solution** is a complete and consistent assignment.

Challenge for today: Given a CSP, find a solution, or show the problem is unsolvable.

Specifying Sudoku

We already have the basic language to specify the Sudoku CSP

X_{ij} : one per empty cell (initial grid might have some numbers)

D_{ij} : $\{1, \dots, 9\}$

Constraints are of the form "some set of cells must all have different values"

Cumbersome to spell out the relation, but this is a very common constraint, so we define:

DEFINITION (*AllDiff*)

The *AllDiff*($X_1, \dots, X_k$) constraint is satisfied exactly when all k variables have different values: $AllDiff(X_1, \dots, X_k) \iff X_i \neq X_j$ for all pairs (i, j) with $1 \leq i < j \leq k$.

Satisfies *AllDiff*(A, B, C)

$(A, B, C) = (1, 3, 2)$

Violates *AllDiff*(A, B, C)

$(A, B, C) = (1, 1, 2)$

Sudoku uses one *AllDiff* constraint for each row, column, and box ($9 + 9 + 9 = 27$ constraints).

What can we infer from *AllDiff*?

Algorithms will reason about constraints to detect failure as early as possible.

Given 5 variables $X_1, \dots, X_5$,
 $D_1 = D_2 = \{1, 2\}$, $D_3 = \{1, 2, 3\}$, $D_4 = \{3, 4\}$, $D_5 = \{4, 5\}$
 $AllDiff(X_1, \dots, X_5)$

Suppose we will branch on X_3 and choose $X_3 = 1$. What can we deduce?

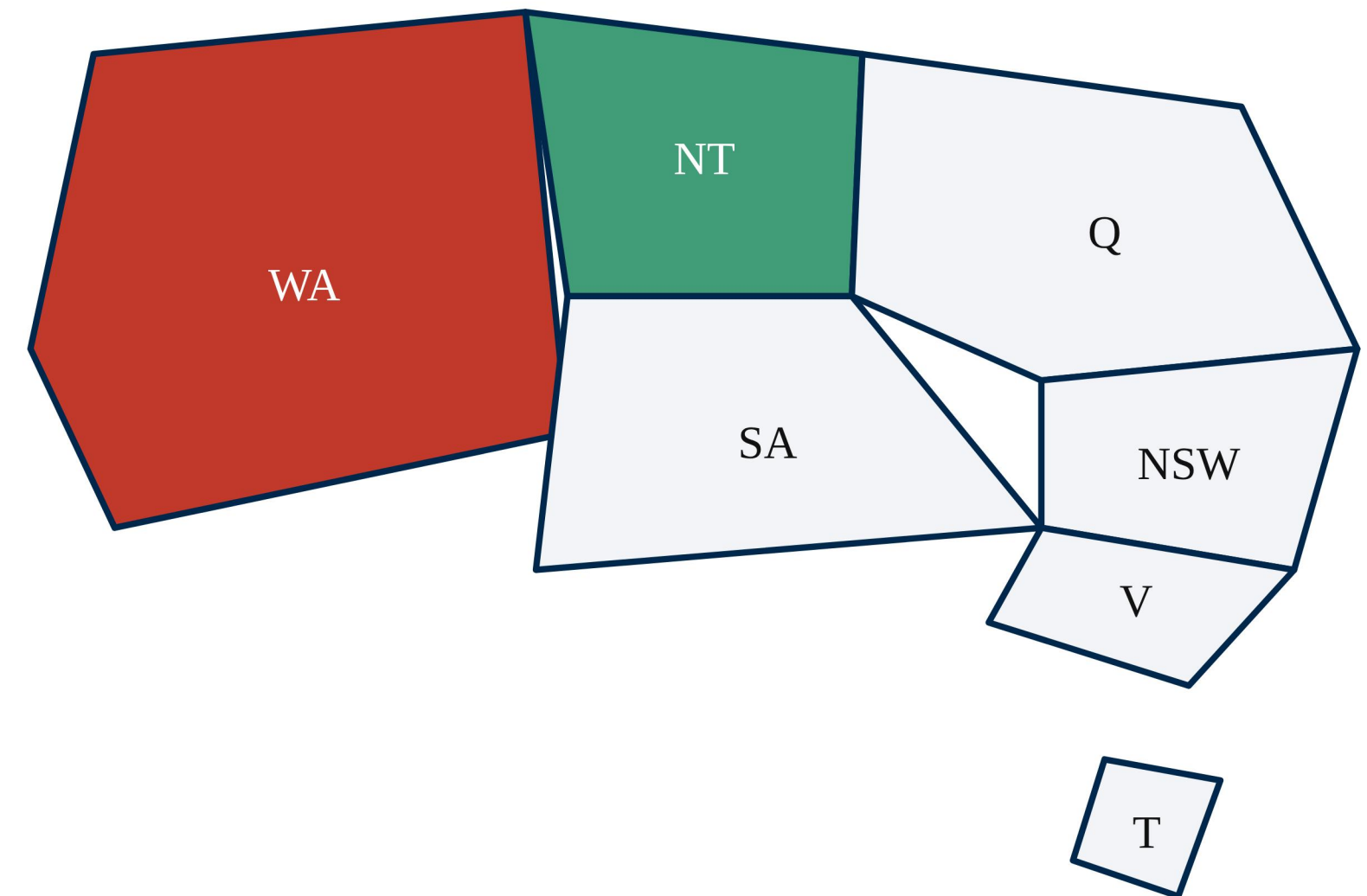
Inference: X_1 has to be assigned 2.

Inference: X_2 also has to be assigned 2.

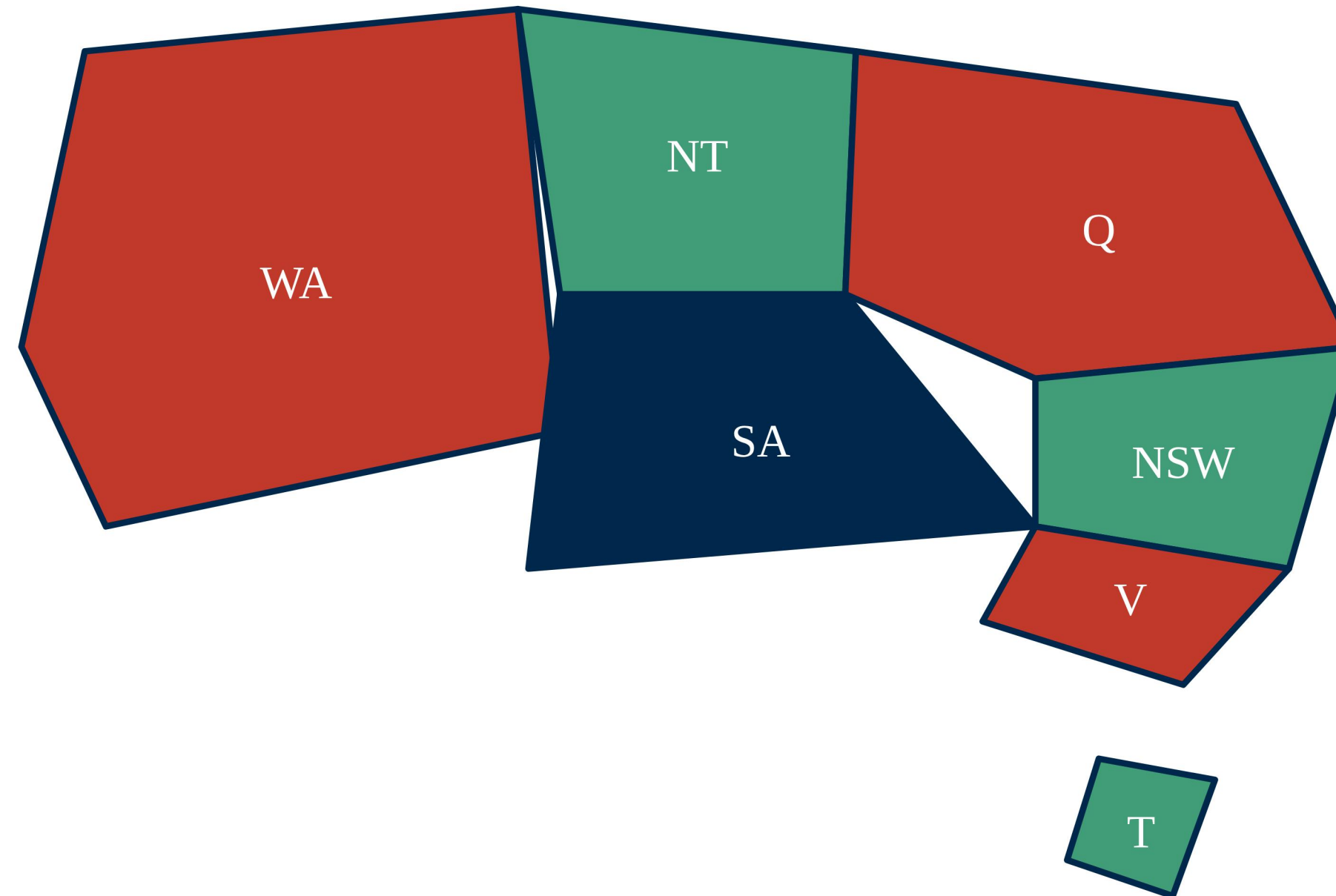
Inference: But this violates *AllDiff*.

Therefore $X_3 = 1$ cannot lead to a solution. Backtrack and try another value.

This process of deriving forced choices is called propagation.



A complete coloring



In CSPs, we only care about the final assignment, not the search path

Example: lecture scheduling

Lecture scheduling CSP: choose room and lecture times to a set of courses.

Rooms: DOW 1010 has 120 seats + recording; DOW 1200 has 60 seats and no recording.

Courses: CSE 592 has 98 students; recording required; instructor unavailable at 2pm.

Model: one variable X_i per course; each value is a (time, room) pair.

Lecture	Initial candidates
CSE 592	(9am, DOW 1010), (9am, DOW 1200) , (2pm, DOW 1010)
CSE 583	(9am, DOW 1010), (11am, DOW 1010)
CSE 598	(9am, DOW 1200), (11am, DOW 1010)

For CSE 592, DOW 1200 violates capacity and recording; 2pm violates instructor availability.

Constraint arity

We can classify constraints based on the size of their scope (how many variables they involve).

Unary	one variable	room supports recording
Binary	two variables	$WA \neq NT$
k-ary	exactly k variables	$A + B + C + D = 2$ ($k = 4$)
Global	More than 2 variables	$AllDiff(X_1, \dots, X_k)$

NOTE

"Global" does not need to involve every variable.

Constraint arity will impact how we implement inferences.

Simplifying our CSP representation

If every constraint is binary, the CSP has a very convenient representation:

Constraint graph

CSP variables as nodes, constraints as edges.

Inference

Look at one edge and two domains at a time.

Convenient, but what if the CSP given to us has unary and global constraints?

Unary constraints are easy: just delete invalid values from the domains

Turns out we can also transform each global constraint into a set of binary constraints (+ new auxiliary variables).

Example: eliminating a 4-ary constraint

Let $A, B, C, D \in \{0, 1\}$, with exactly two variables equal to 1:

$$A + B + C + D = 2$$

The constraint allows exactly six (A, B, C, D) tuples:

$$\begin{array}{ccc} (1, 1, 0, 0) & (1, 0, 1, 0) & (1, 0, 0, 1) \\ (0, 1, 1, 0) & (0, 1, 0, 1) & (0, 0, 1, 1) \end{array}$$

Key idea: add one auxiliary variable U_{ABCD}

Domain

The set of 6 tuples above

Constraints

One binary constraint with each of A, B, C, D enforcing agreement between that variable and the tuple.

Example: eliminating a 4-ary constraint

More precisely:

$$U \in \{(a, b, c, d) \in \{0, 1\}^4 : a + b + c + d = 2\}$$

Meaning Assigning $U = (1, 0, 1, 0)$ will force $A = 1$, $B = 0$, $C = 1$, and $D = 0$.

Four binary constraints require each variable to match its entry in U :

$$U[1] = A \quad U[2] = B \quad U[3] = C \quad U[4] = D$$

One tuple variable + four binary constraints replace the previous 4-ary constraint

The general hidden-variable encoding

This is a general strategy to eliminate any global constraint

For a general constraint $C(X_1, \dots, X_k)$:

1. Let rel_C be the set of tuples satisfying C .
2. Add a variable U_C with domain $D(U_C) = rel_C$.
3. For each X_i , add the binary constraint $U_C[i] = X_i$.

$$U_C[1] = X_1 \quad \dots \quad U_C[k] = X_k$$

Any global constraint can be represented using only binary constraints

This drastically simplifies algorithm design for a CSP solver.

CAREFUL

The tuple domain can be very large. Some applications might be better off without this reduction.

CONSTRAINT PROPAGATION

Search or inference?

RECALL

Last lecture, search made progress by repeatedly selecting a next node to expand.

In a CSP, we have two ways to make progress:

Search

Expand and follow one branch

$$X = red \text{ or } X = green$$

Inference

Rule out possibilities using constraints

$$X = red, X \neq Y \Rightarrow red \notin D(Y)$$

Inference simplifies our choices while preserving solutions

Domains are our current knowledge

Our current knowledge about the solution is expressed in the domains.

Before

$$D(X) = \boxed{2} \boxed{7}$$

Learn

Another cell in X 's column is fixed to 7

After

$$D(X) = \boxed{7} \boxed{2}$$

constraints + domains $\implies$ smaller domains

A **fixed point** is reached when our inferences stop changing domains.

At that point, we will need to make a decision to make progress.

Local consistency

Suppose we will make a decision on the value of some X_i

Ideally, we'd ask "what values of X_i can we extend to a full solution"?

If we knew, CSPs would be easy!

Instead, we can reason about cheaper **local consistency** conditions:

- **node:** which values satisfy unary constraints on X_i ?
- **arc:** which values can possibly satisfy binary constraints involving X_i ?
- **path:** which values can possibly satisfy two constraints on a path to some third variable?
- **k -consistency:** generalization of path consistency to more than 2 edges

Necessary conditions: if we violate these, we violate global consistency too

Not sufficient conditions. But still useful to reduce choices

We'll look at node and arc consistency.

Node consistency

DEFINITION (NODE CONSISTENCY)

A variable X_i is **node-consistent** when every value in D_i satisfies every unary constraint on X_i : $\forall x \in D_i, \forall C_k$ with $scope_k = \{X_i\} : C_k(x)$.

Example: suppose $D_{CSE592} =$

9am, DOW 1010

9am, DOW 1200

2pm, DOW 1010

98 students, recording required; instructor unavailable at 2pm:

9am, DOW 1010

~~9am, DOW 1200~~

~~2pm, DOW 1010~~

Enforcing node consistency amounts to deleting what we can from domains

Can be done as a cheap pre-processing step to the CSP

DEFINITION (ARC CONSISTENCY)

A directed edge $X_i \rightarrow X_j$ is called **arc-consistent** if every value for X_i has a supporting value for X_j :

$$\forall x \in D_i, \exists y \in D_j : C_{ij}(x, y).$$

$y \in D_j$ is a **support** for $x \in D_i$ when $C_{ij}(x, y)$ holds.

If x has no support, we can safely delete it from D_i .

⚠ CAREFUL

Direction matters: even if $X_i \rightarrow X_j$ is arc-consistent, $X_j \rightarrow X_i$ might not be.

Arc consistency: supports

$$X + Y = 5$$

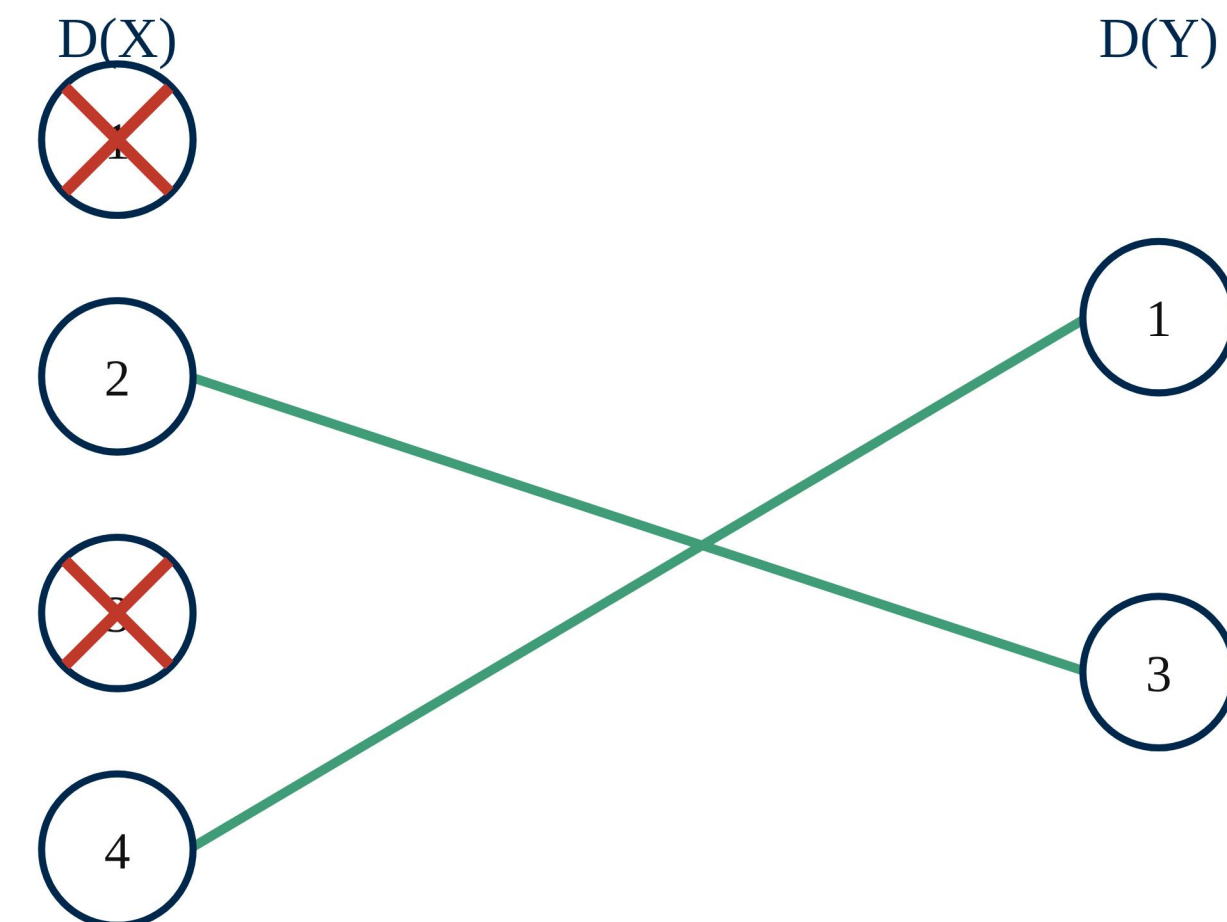
$$D(X) = \{1, 2, 3, 4\}$$

$$D(Y) = \{1, 3\}$$

Which values for X are possible?

- $X = 1$: **no support in Y**
- $X = 2$: supported by $Y = 3$
- $X = 3$: **no support in Y**
- $X = 4$: supported by $Y = 1$

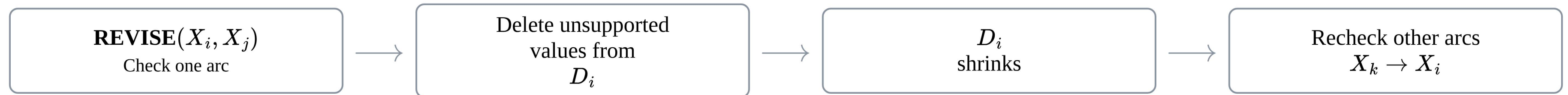
$$D(X) \leftarrow \{2, 4\}$$



From one arc to a fixed point

Let's define a $\text{REVISE}(X_i, X_j)$ operation that reduces the domain of X_i by using arc-consistency

 **OBSERVATION**
Reducing X_i might allow us to reduce choices for other X_k that are not supported in X_i anymore



The **AC-3** algorithm repeats the procedure above until a fixed point.

Arcs that need to be re-checked are kept into a queue.

Two conditions can make AC-3 stop:

A domain becomes empty

No solution extends the current assignment

The queue becomes empty

Arc-consistent fixed point

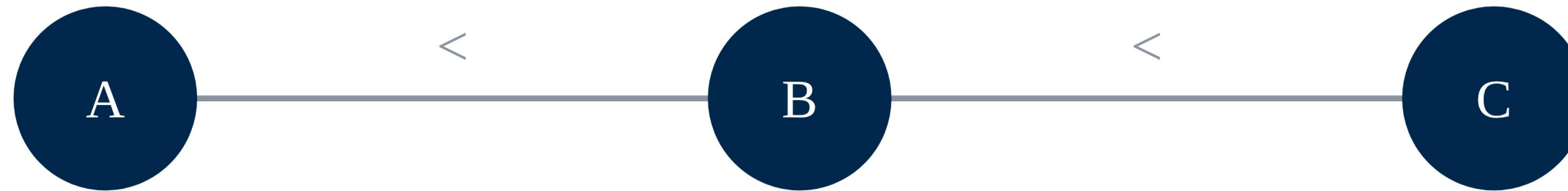
The REVISE operation

```
1  function REVISE(csp, Xi, Xj):
2      revised = false
3      for x in Di:
4          if no y in Dj satisfies Cij(x, y):
5              delete x from Di
6              revised = true
7      return revised
```

Cost: $O(d^2)$ for maximum domain size d , assuming each constraint test costs $O(1)$.

```
1  function AC-3(csp):
2      queue = all directed arcs (Xi, Xj)
3      while queue is not empty:
4          (Xi, Xj) = queue.pop()
5          if REVISE(csp, Xi, Xj):
6              if Di is empty: return failure
7              for Xk in NEIGHBORS(Xi) - {Xj}:
8                  queue.add((Xk, Xi))
9      return success
```


AC-3: propagation example



$D(A) = \{1, 2, 3\}$

$D(B) = \{1, 2, 3\}$

$D(C) = \{1, 2, 3\}$

Revise	Values removed	Updated domain
$C \rightarrow B$	1 from D_C	$D_C = \{2, 3\}$
$B \rightarrow A$	1 from D_B	$D_B = \{2, 3\}$
$B \rightarrow C$	3 from D_B	$D_B = \{2\}$
$A \rightarrow B$	2, 3 from D_A	$D_A = \{1\}$
$C \rightarrow B$	2 from D_C	$D_C = \{3\}$

Here, AC-3 deduces $A = 1$, $B = 2$, $C = 3$ **without search**

What does AC-3 guarantee?

When AC-3 is done, one of two conditions has to be true:

Some $D_i = \emptyset$

No solution extends the current assignment

or

Every arc is consistent

The same solutions remain, with no unsupported values

c binary constraints; maximum domain size d

One REVISE	$O(d^2)$
Directed arcs	$2c = O(c)$
Rechecks per arc	at most d after domain deletions

Time: $O(cd^3)$

Queue space: $O(c)$

Usually much cheaper than exponential search

Local does not imply global

Of course, arc-consistency is necessary but not yet sufficient

Let's look at a simple map coloring example

$$D_A = D_B = D_C = \{blue, red\}$$

Constraint: $X \neq Y$ for each pair

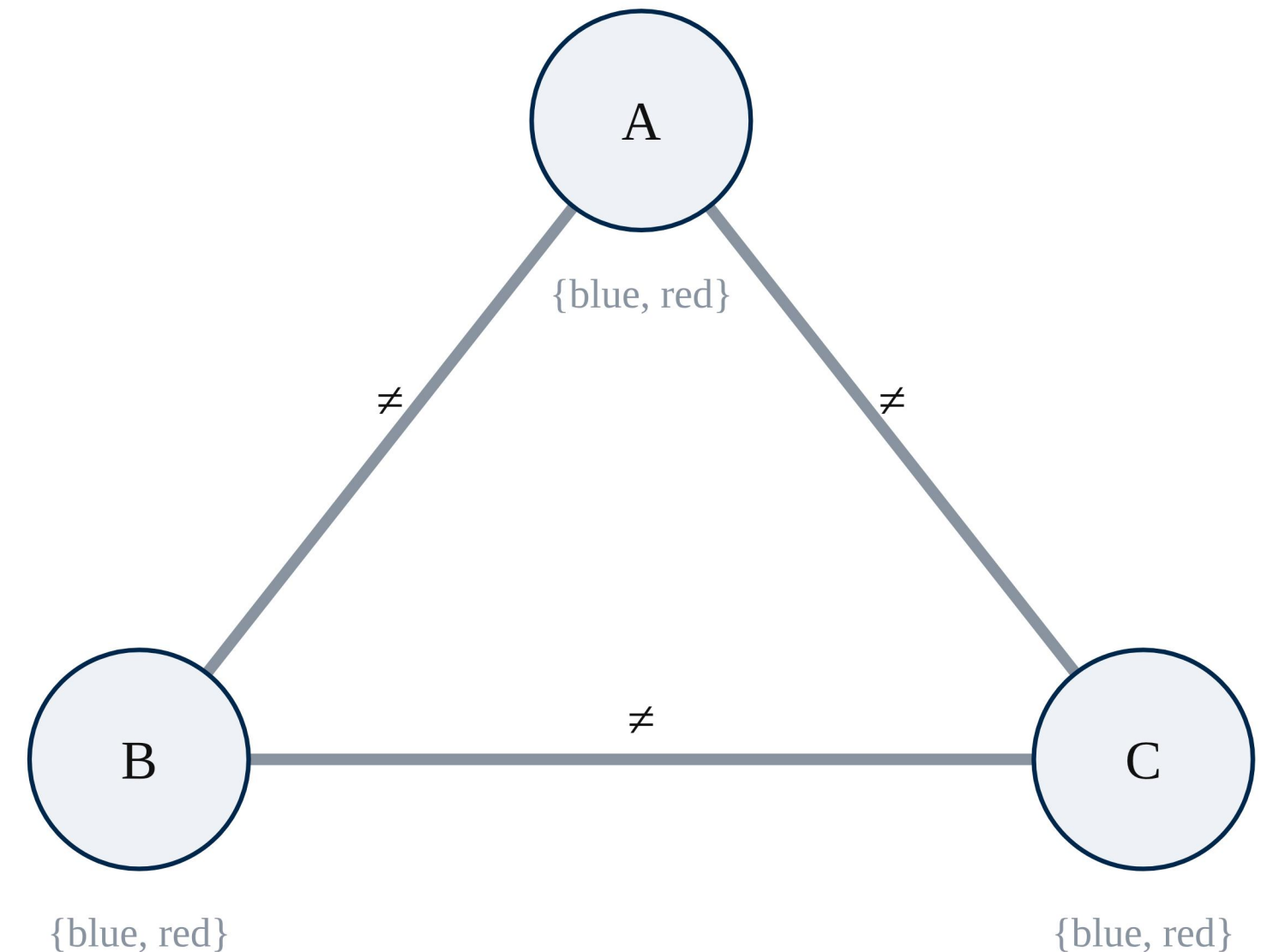
Arc-consistent? Yes.

Solvable? No.

On each edge, either color supports the other color.

But three mutually different variables cannot use only two colors.

We'll only be able to see this after making a choice on some variable.



BACKTRACKING SEARCH

Backtracking search

General backtracking search will look like:

1. Choose an unassigned variable
2. Choose one value to try
3. Infer consequences
4. If some domain is empty: branch failed, undo and try again
5. Otherwise, recurse

This is depth-first search over partial assignments, interleaving with inference.

Avoiding redundant search

A naive search action can choose both:

- which variable $x \in X$
- which value $v \in D_x$



OBSERVATION (COMMUTATIVITY)

Assigning distinct variables in either order produces the same partial assignments: $(X = a; Y = b)$ is equivalent to $(Y = b; X = a)$.

Therefore, we can choose one variable at a time and branch only on its values.

If all values for some X_i fail, we **do not need to try other starting with other variables!**

This lets us avoid an $n!$ time complexity factor in the worst case

Backtracking search: pseudocode

```
1  function BACKTRACK(csp, assignment):
2      if assignment is complete: return assignment
3      X = SELECT-UNASSIGNED-VARIABLE(csp, assignment)
4      for v in ORDER-DOMAIN-VALUES(X, csp, assignment):
5          if v is consistent with assignment:
6              assign X = v
7              inferences = INFERENCE(csp, X, v)
8              if inferences != failure:
9                  result = BACKTRACK(csp, assignment)
10                 if result != failure: return result
11             undo X = v and inferences
12     return failure
```

Three choices in backtracking

Variable Which variable next?

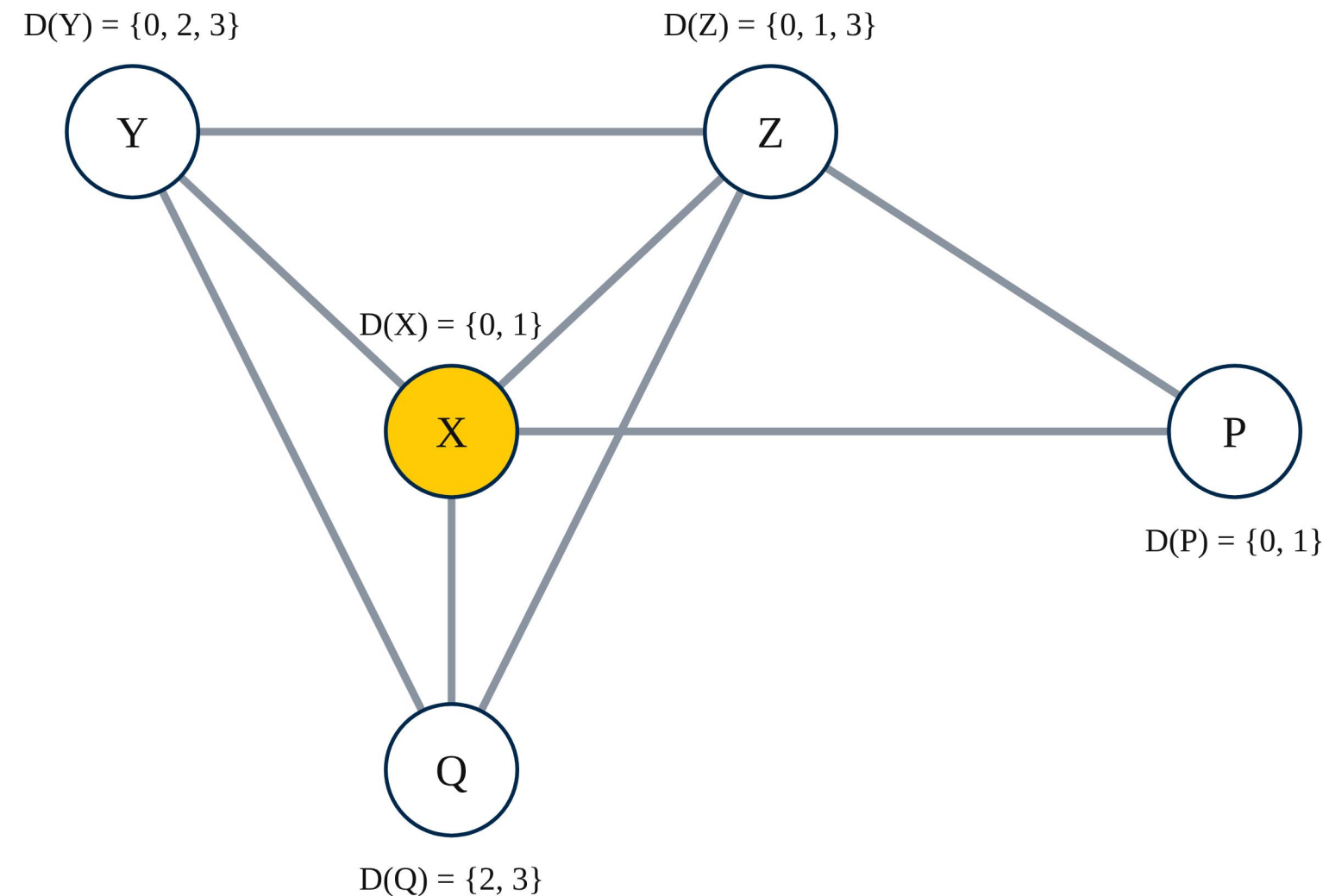
Value Which value first?

Inference How much propagation?

Several domain-general heuristics available for each

Where should we branch?

A list-coloring CSP with colors $\{0, 1, 2, 3\}$:



Every edge means its endpoints must have different colors.

AC-3 here removes nothing, so we need to branch.

Q: which variable should we branch on first?

Say we choose $Y = 0$. If that fails, that rules out $\frac{1}{3}$ of the assignments

Say we choose $X = 0$. If that fails, that rules out $\frac{1}{2}$ of the assignments

Intuition: choose most constrained variable first

Example: if some V has only 1 value left, no reason to not assign it and propagate!

Minimum remaining values

DEFINITION (MRV)

The **minimum remaining values** (MRV) heuristic chooses the unassigned variable X_i with the fewest remaining values in its domain D_i .

What happens if there is a tie?

We're trying to gain as much information as possible about currently unassigned variables

Constraints are what give us information (besides search)

This leads to the intuition of choosing the variable involved in more constraints with other unassigned variables.

Example: if a variable is completely free (no constraints), deciding it gives us no further information!

DEFINITION (DEGREE HEURISTIC)

When MRV ties, choose the variable involved in the most constraints with other **unassigned** variables.

Which value should come first?

Say we decide to branch on X

If there is no solution, value order won't matter: we will eventually try all of them and fail

But if there is a solution, order matters since we stop as soon as we find a solution

So, one intuition is: try **values more likely to work first**

Example: suppose X must differ from A , B and C , with:

$D(A)$
 $\{red, green\}$

$D(B)$
 $\{red, blue\}$

$D(C)$
 $\{red, yellow\}$

Try $X = red$

Remove *red* from all three neighbor domains.

Try $X = green$

Remove *green* only from $D(A)$.

green preserves more choices, so intuitively more likely to work

Least constraining value

That is the intuition for the LCV heuristic:

DEFINITION (LCV)

The **least constraining value** (LCV) heuristic first tries the value $v \in D_i$ that removes the fewest choices from neighboring unassigned variables.

How do we compute "how many choices $X_i = v$ removes"?

Stronger but more expensive: AC-3

Weaker but cheaper: just look at neighbors

DEFINITION (FORWARD CHECKING)

After assigning $X = v$, **forward checking** removes from every unassigned neighbor Y each value inconsistent with $X = v$:
 $D_Y \leftarrow \{y \in D_Y : C_{XY}(v, y)\}$. It fails if any $D_Y = \emptyset$.

Maintaining arc consistency

If we do want to always be arc consistency, we do not need to start with a full queue each time!

DEFINITION (MAC)

Maintaining arc consistency (MAC) restores arc consistency after every choice: assign $X = v$, set $D_X = \{v\}$, then run AC-3 beginning with only the affected arcs.

Forward checking

Less inference at each node;
usually explores more nodes

MAC

More inference at each node;
often explores fewer nodes

New assignment $\Rightarrow$ new information $\Rightarrow$ propagate again

Beyond chronological backtracking

A failure may depend on older choices, not the last ones.

More clever algorithms can perform non-chronological backtracking:

Whenever we reach a conflict, (1) find its root cause (subset variables that mattered), (2) backtrack higher by skipping unrelated decisions

Example: after deciding $X_1 = 1, X_2 = 1, X_3 = 1, \dots, X_{99}$, we might eventually realize that $X_2 = 1, X_3 = 1, X_{100} = \text{anything}$ always fails (e.g., leaves no values for X_{101})

If we identify this is the root cause, we can go straight to undo $X_3 = 1$

We can also **learn** this as a new constraint: $X_2 \neq 1$ **or** $X_3 \neq 1$

If we eventually backtrack to X_1 we can avoid repeating this mistake.

This is the idea of backtracking with *no-goods* ($X_2 = 1, X_3 = 1$ is no good!).

Our approach today: "global search". Maintain and update all domains.

Another approach is **local search**: just keep one solution.

Solution might be invalid, but we can try local changes (e.g., changing one variable)

If we eventually fix all violations, we succeed

Book goes in depth into many local search methods

Many new avenues in recent research. For example, we can train a Transformer to predict these improvement steps!

Self-Supervised Transformers as Iterative Solution Improvers for Constraint Satisfaction

Yudong Xu, Wenhao Li, Scott Sanner, Elias Boutros Khalil — International Conference on Machine Learning, 2025

Their model sometimes generalizes even to unseen CSPs!

Analogous to learning search heuristics we mentioned last lecture.

Some other papers used CSP solvers to generate synthetic reasoning data for LLMs (the reasoning strategies we've seen today are broadly useful beyond CSPs!)

CSPs

Variables + domains + constraints: much more structure in the state than atomic states.

Inference by local consistency: Arc consistency + AC-3

Every value needs support across each outgoing arc. AC-3 revises and propagates through affected arcs until failure or a fixed point. Path consistency and k -consistency go beyond single arcs.

Ordering: Choosing a variable

MRV selects the fewest remaining values; degree heuristic breaks ties by choosing the variable with the most unassigned neighbors.

Ordering: Choosing a value

LCV preserves neighbor choices. Forward checking looks at immediate neighbors; MAC restores arc consistency after each assignment.

Combinations of these ideas (+ others like constraint learning) allow modern CSP solvers to scale to 10k-100k variables, millions of constraints

Next lectures: formal logic

We will look at an even more structured problems, reasoning in formal logic.

Next lecture: propositional logic

Variables

Atomic logical propositions: P , Q , R , $\dots$

Domains

Truth values $\{false, true\}$

Constraints

Logical formulas over those propositions

Then, first-order logic, where our language will get much richer